LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-PROC-420704

High Level Trigger Configuration and Handling of Trigger Tables in the CMS Filter Farm

G. Bauer, U. Behrens, V. Boyer, J. Branson, A. Brett, E. Cano, A. Carboni, M. Ciganek, S. Cittolin, V. O'dell, S. Erhan, D. Gigi, F. Glege, R. Gomez-Reino, M. Gulmini, J. Gutleber, J. Hollar, D. Lange, J. C. Kim, M. Klute, E. Lipeles, J. A. L. Perez, G. Maron, F. Meijers, E. Meschi, R. Moser, E. G. Mlot, S. Murray, A. Oh, L. Orsini, C. Paus, A. Petrucci, M. Pieri, L. Pollet, A. Racz, H. Sakulin, M. Sani, P. Schieferdecker, C. Schwick, K. Sumorok, I. Suzuki, D. Tsirigkas, J. Varela

November 24, 2009

International Conference on Computing in High Energy and Nuclear Physics
Victoria, Canada
September 2, 2007 through September 7, 2007

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

High Level Trigger Configuration and Handling of Trigger Tables in the CMS Filter Farm

Gerry Bauer[11], Ulf Behrens[2], Vincent Boyer[6], James Branson[7], Angela Brett[6], Eric Cano[6], Andrea Carboni[6], Marek Ciganek[6], Sergio Cittolin[6], Vivian O'dell[9], Samim Erhan[6,8], Dominique Gigi[6], Frank Glege[6], Robert Gomez-Reino[6], Michele Gulmini[6], Johannes Gutleber[6], Jonathan Hollar[10], David Lange[10], Jin Cheol Kim[5], Markus Klute[11], Elliot Lipeles[7], Juan Antonio Lopez Perez[6], Gaetano Maron[3], Frans Meijers[6], Emilio Meschi [6,12], Roland Moser [1,6], Esteban Gutierrez Mlot[6], Steven Murray[9], Alexander Oh[6], Luciano Orsini[6], Christoph Paus[11], Andrea Petrucci[3], Marco Pieri[7], Lucien Pollet[6], Attila Racz[6], Hannes Sakulin[6], Matteo Sani[7], Philipp Schieferdecker[6], Christoph Schwick[6], Konstanty Sumorok[11], Ichiro Suzuki[9], Dimitrios Tsirigkas[6], Joao Varela[4,6]

1. Vienna University of Technology, Vienna, Austria
2. DESY, Hamburg, Germany
3. INFN - Laboratori Nazionali di Legnaro, Legnaro, Italy
4. LIP, Lisbon, Portugal
5. Kyungpook National University, Daegu, Kyungpook, South Korea
6. CERN, Geneva, Switzerland
7. University of California, San Diego, La Jolla, California, USA
8. University of California, Los Angeles, Los Angeles, California, USA
9. FNAL, Batavia, Illinois, USA
10. LLNL, Livermore, California, USA
11. Massachusetts Institute of Technology, Cambridge, Massachussetts, USA

Abstract. The CMS experiment at the CERN Large Hadron Collider is currently being commissioned and is scheduled to collect the first pp collision data in 2008. CMS features a two-level trigger system. The Level-1 trigger, based on custom hardware, is designed to reduce the collision rate of 40 MHz to approximately 100 kHz. Data for events accepted by the Level-1 trigger are read out and assembled by an Event Builder. The High Level Trigger (HLT) employs a set of sophisticated software algorithms, to analyze the complete event information, and further reduce the accepted event rate for permanent storage and analysis. This paper describes the design and implementation of the HLT Configuration Management system. First experiences with commissioning of the HLT system are also reported.

1. Introduction

The CMS [1] High Level Trigger (HLT) [2] is designed to reduce the Level-1 accept rate of 100 kHz by about a factor 1000 by analyzing full-granularity detector data using software reconstruction and

[12] Corresponding author: Emilio Meschi, CERN, CH1211, Geneva 23, Switzerland, E-mail: emilio.meschi@cern.ch

filtering algorithms developed within the same reconstruction framework employed for offline reconstruction and analysis [3]. The HLT runs on a computing farm consisting of standard CPU (Filter Farm). The Filter Farm is part of the Event Filter complex. The Event Filter architecture and data flow are schematically depicted in figure 1.

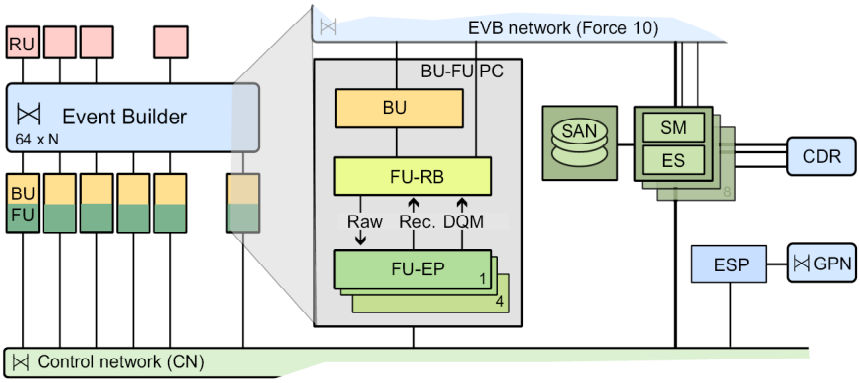

Figure 1. Architecture and data flow of the CMS Event Filter

The Event Filter complex

- Collects events accepted by the Level-1 Trigger system from the Event Builder and distributes them to worker nodes in the Filter Farm for further processing
- Performs basic consistency checks on the event data
- Runs the HLT to process and select events for storage
- Generates, collects, and distributes Data Quality Monitoring (DQM) information resulting from online event processing in the HLT
- Serves a subset of the events to local and remote online consumers (EventServer, ES) for calibration and DQM
- Routes selected events to local storage in several online streams according to their trigger configuration (Storage Manager, SM)
- Transfers data from local storage at the CMS site to mass storage in the CERN data centre at the Meyrin site via the Central Data Recording (CDR)

The Event Filter hardware consists of a large farm of about 2500 worker nodes, hosting the HLT execution (Filter Farm), and a data logging system connected to a storage area network (SAN). The worker nodes form the receiving end of the second of two stages of event building. A logically separate switch fabric provides the connectivity to a set of 8 data logging nodes. These data logging nodes are connected to a Fibre-Channel SAN, which is capable of a peak bandwidth of 1 GB/s and has a capacity of several 100 TB.

The farm is logically subdivided into groups of processing nodes (Builder/Filter Unit, BUFU). Each BUFU hosts an identical set of software modules in a distributed environment based on the CMS online framework, XDAQ [4].

Two separate applications, the Filter Unit "Resource Broker" (RB) and the Event Processor (EP), provide the HLT functionality. The RB is responsible for managing memory resources for input and output to/from the HLT processes, and the communication with the Event Builder and the data logger. A full event is handed by the RB, upon request, to the Event Processor (EP). The EP uses the CMS reconstruction framework to steer the execution of reconstruction and selection code forming the HLT selection. Multiple EP processes can coexist in a single worker node to provide concurrent execution and thus saturate the worker node CPU resources.

The HLT algorithms are configured at the start of each data taking run, under supervision of the Run Control system [5]. The HLT configurations (trigger tables) are created and managed within the HLT configuration management system. The creation of HLT configurations, consisting of many software

modules organized in O(100) trigger paths, and their deployment into the distributed online environment, are centered around a carefully designed database schema abstracting the features of the algorithms and their organization in a trigger table. The HLT configuration management system is the subject of the following sections.

2. Design considerations

The use of a unique reconstruction framework for both offline and online execution of reconstruction and filtering algorithms provides a high degree of flexibility and sophistication, while enabling a wide community of developers and physicists to efficiently contribute to the development, testing, and validation of the HLT algorithms. It also guarantees fast migration of offline reconstruction and selection algorithms as more effective strategies are developed, when new physics requirements, machine or detector conditions, and available resources warrant.

For online execution in a distributed environment, the reconstruction framework must be encapsulated in the online framework. The latter is responsible, among other, of delivering configuration information and guaranteeing its consistency across the entire distributed system.

2.1. HLT in the CMS Reconstruction Framework

The CMS reconstruction framework uses a modular architecture, enforcing component interfaces for the implementation of the different functionalities: input, reconstruction, filtering, access to non-event data, other services (such as logging, monitoring, etc.), and output. Algorithms communicate exclusively via a single event data bus where reconstruction products and other event-based information is placed. The event data model enables persistent storage of any algorithm by-product, which can later be uniquely identified and traced.

Modules are loaded, instantiated, and configured using a plug-in mechanism at initialization time, according to an object representation of the configuration, constructed by parsing a configuration document. The configuration objects map to unique instances of the modules and completely define their behaviour by setting all relevant parameters. The configuration also specifies a sequence of execution of the modules.

Different configuration grammars have been developed to map the object model and provide flexible and powerful file-based configuration for offline jobs.

In the HLT, module instances providing the event selection must be arranged in “trigger paths”. An HLT path consists of a sequence of reconstruction and filter modules of increasing complexity and sophistication, designed to identify a certain category of events. The same algorithm instance can be used in multiple paths. The framework guarantees that the same algorithm is only run once for a given event. At different stages in the path, filter modules examine the event content for specific features. The path execution is interrupted in case of a negative filter decision. This way, time-consuming algorithms at the end of a path are only seldom executed. Only events surviving at least one complete path are accepted for permanent storage[13].

To guarantee reproducibility of the results, and the complete traceability of events accepted by the HLT, strict scheduling of the execution of HLT is imposed. Every HLT path is executed in a predetermined sequence and all paths are executed till the first reject decision or to the end.

A typical offline workflow for development and study of HLT algorithms and table is illustrated in figure 2. An algorithm is known to the framework by its assigned plug-in module name and its parameterization (left), which are completely defined in the source code. When the algorithm is used in a particular sequence, defined in a dedicated subdocument, a module *instance* is defined with a list of non-default parameter values. A single instance of a module can be used in different sequences and multiple instances of the same algorithm can be defined with different parameters.

---

[13] Special provisions are made to accept a small unbiased subset of otherwise rejected events for diagnostic purposes

The various subdocuments must finally be merged into a single menu to be fed to an offline job for execution.

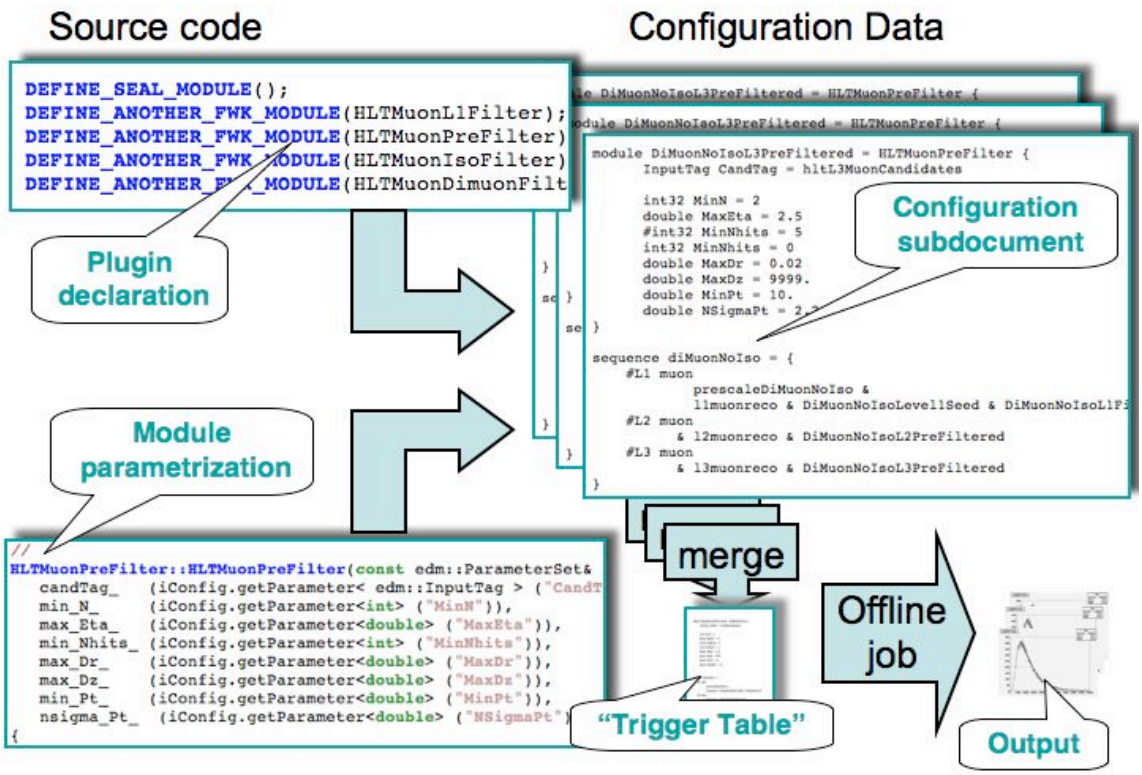

Figure 2. Schematic view of the HLT offline workflow using file-based configuration

2.2. HLT in the Online systems

The XDAQ framework provides a single 'executive' program to load online application components, and provides services to dispatch configuration and control messages, and give access to communication and data transport in the distributed system. Applications consist of loadable modules using the XDAQ services. The offline framework encapsulation is achieved in the EP by enabling the online application to load offline framework modules, and defining special interfaces for event data input/output, control, and monitoring.

The Run Control system is responsible for the deployment of the various applications in the distributed environment, their configuration, and the control of the distributed system using the aforementioned interfaces. The latter is realized by a tree of state machines (Function Managers, FM), which keep track of the overall state of the experiment. The HLT Supervisor (HLTS) is the Function Manager that controls the configuration and execution of the HLT inside the EP. In particular, the HLTS receives from Run Control, at configuration time, the instruction to load a particular HLT table in the form of a symbolic key. This symbolic key can be interpreted by the HLTS in different ways depending on the source of the configuration information, but the corresponding configuration must finally be delivered to all EP processes in the form of a document that can be parsed by the reconstruction framework to produce a valid HLT executable.

A file-based configuration system can be cumbersome to maintain. Concurrent versioning of multiple tables quickly becomes a technical challenge. The evolution of grammars and tools for configuration

manipulation, and of the underlying algorithms to be configured, presents obvious problems for long-term maintenance of configuration data. The evolution of the algorithms across different versions of the code, in particular, requires constant maintenance even of functionally identical tables. In an online environment, where concurrent access to file systems by many processes must be supported, guaranteeing data consistency can also pose problems. Using a database to store and catalogue CLOBs only partially solves the problem.

2.3. Requirements

Based on the above considerations, the following requirements where identified for a db-oriented HLT Configuration Management System:

- The configuration database schema should provide an abstract model of the framework components and their assembly into a trigger table, with little or no dependence on the configuration grammar
- The management system should provide the ability to populate a configuration based on a particular version of the underlying code, and the possibility of subsequently migrating a configuration to a different version
- A graphical tool should be provided to populate HLT tables. This tool should enforce basic rules imposed by the reconstruction framework, and basic consistency of the configuration information
- The configuration system should support the basic online workflow to deploy a particular table for data-taking. Tables should be identified by a single symbolic key. The system should take care of reformatting the table into a valid configuration document which can be parsed by the reconstruction framework
- The configuration system should support offline development and validation of HLT algorithms. Convenient offline access to existing tables, and the possibility of extracting configuration documents for offline use, as well as intuitive traversal of large tables, should also be provided. The migration of offline-validated HLT tables to the online environment should be as transparent as possible.
- Referential integrity and data consistency are expected to be guaranteed by this system across the entire lifetime of the experiment.

3. Configuration System Design and Implementation

The HLT database-driven configuration system, HLTConfDB, schematically drawn in figure 3, consists of four main components. A relational database schema, a code parser, a graphical user interface, and a converter to generate configuration documents for use in the reconstruction framework.

3.1. Database

A relational database schema is defined to describe the relations between paths, services, modules, and their parameters. The schema is maintained for both Oracle and MySQL. Each HLT configuration in the database is related to one specific release of the CMS software. The algorithms and modules provided in that release are the building blocks of each configuration. These building blocks are stored as templates in the database. A module template contains the name by which it is identified to the framework, and a relation to a particular software version. It also contains references to a set of parameters that represent its default parameter set as defined in the code itself. A configuration is stored as a set of paths, services, and global parameters. A path is a sequence of module instances (algorithms) where a given instance can be used multiple times inside different paths. A module instance consists of an instance name (label), a reference to its module template, and a set of parameters that differ from their default. The module label uniquely identifies the products of the algorithm execution in the event data.

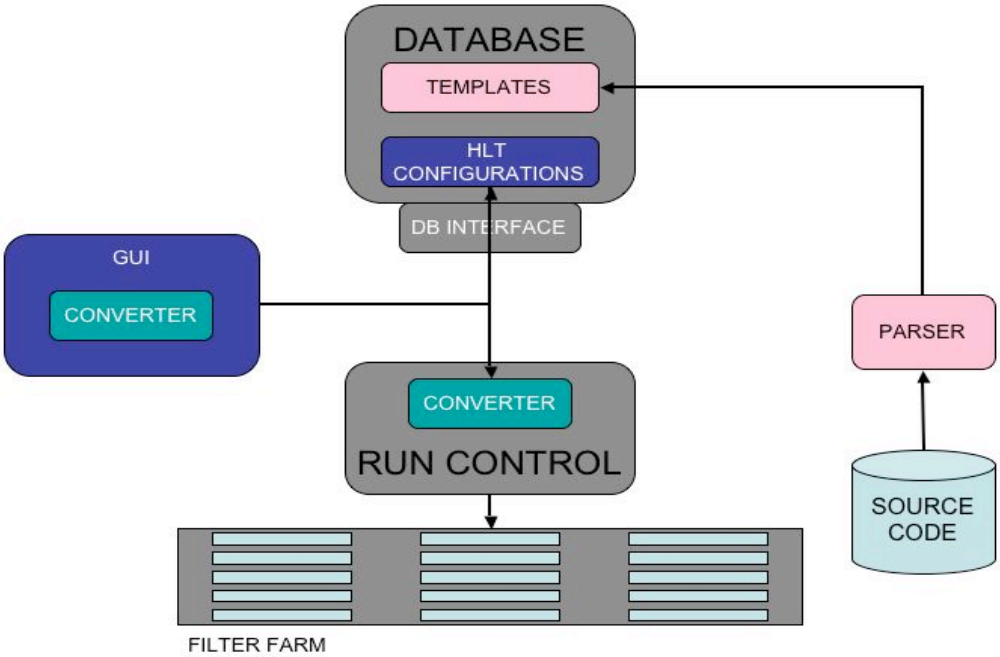

Figure 3. Schematic view of the configuration system and its interactions

3.2. Parser

The parser consists of a set of python scripts that traverse the complete source tree of a given reconstruction software release, and discover all available modules, their type (reconstruction or filter algorithm, service, etc.) and, if feasible, all configuration parameters and their default values (see left of figure 2 for an example). The discovered software components are stored in the database as templates; when one of these is used in a configuration, an instance of a template is created and related to its parent template.

The current implementation of the parser is capable of traversing the entire source tree of the CMS software, consisting of over 1M SLOCs, in about 1min, discovering almost 1000 components, with their over 5000 parameters, and storing the corresponding templates. It produces a report listing modules with problems, which is usually indication of bad or obscure coding practices.

3.3. Graphical User Interface

The HLTConfDB graphical user interface provides tools to generate an HLT configuration based on templates stored in the database for a given software release. When a configuration is created, the corresponding template information is loaded from the database. The java-based GUI displays the configuration in a tree like structure. Context menus, filled from the template information, enable the addition of paths to a configuration tree, of modules to paths, and the modification of parameters of a module instance. The creation of a valid configuration according to basic rules of the software framework is enforced. For example, a configuration containing a module instance with a mandatory parameter without a default, or containing two different instances of the same module with identical parameter values, cannot be saved. Whenever a configuration is saved to the database, all previous versions are kept and a new unique key is assigned to the new version. The GUI has tools to support drag-and-drop of fragments across two different configurations, migration of entire tables across databases, and update of a table to a different software version. It integrates the Converter tool (see

next section) for quick export of a configuration to a specific grammar or format. A locking mechanism is provided in the database to prevent concurrent editing of the same configuration.
A snapshot of the HLTConfDB GUI is illustrated in figure 4.

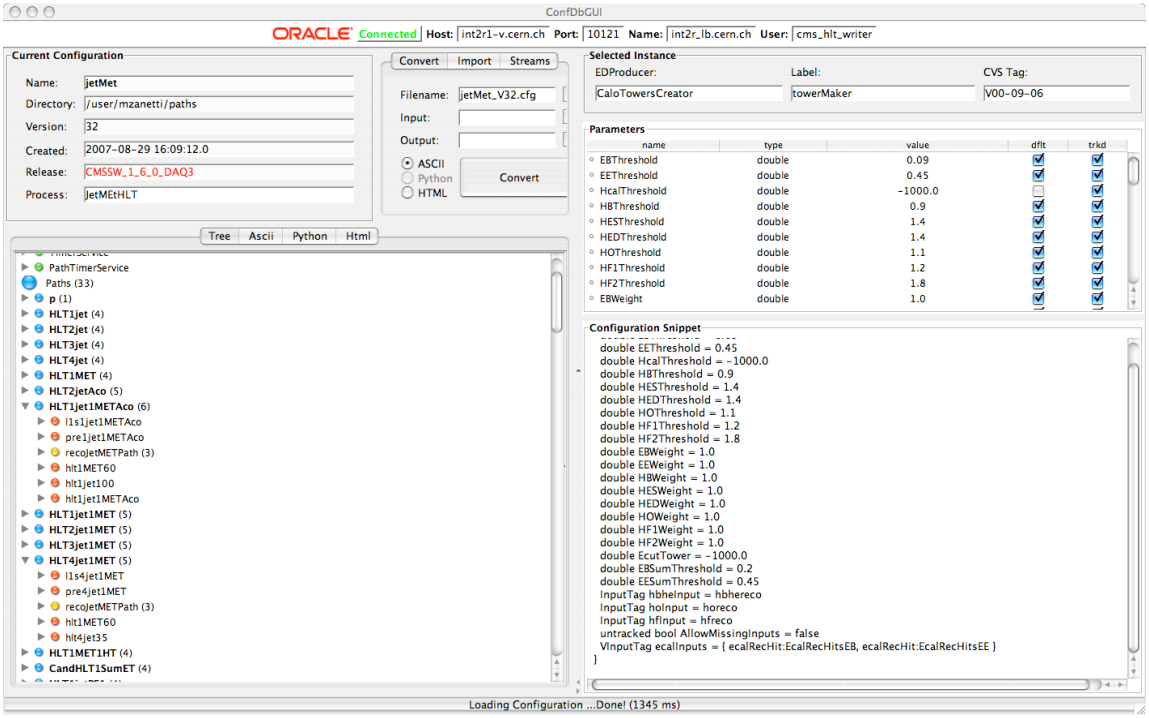

Figure 4. Screenshot of the ConfDB GUI

3.4. Converter

Once configurations are stored in relational database tables, the Converter accesses the database using the same database API as the GUI, to retrieve a given configuration and transform the information into other formats: a valid job configuration document, a webpage, etc. The Converter java API provides implementations for each useful representation of a configuration. Currently these include the initial text-based configuration grammar implemented for the offline reconstruction [3], the new Python-based configuration, and an HTML representation, which can be useful for quick access to key information about currently running HLT tables. Run Control uses the Converter API, in the HLTS, to retrieve the HLT configuration to be deployed in the Filter Farm, based on a key provided by the operator, which can be selected among the valid configurations according to the need. Web access to the HLTConfDB information is provided via a servlet using the Converter API from within an Apache tomcat servlet engine. This can be used by developers wishing to test an existing configuration, physicists interested in a detail of the HLT configuration, etc.

4. Operation

4.1. Offline and Online Workflows

The HLT configuration management system is designed to support not only online operation, but also offline development and validation of trigger selections, and the integration of the trigger table. Figure 5 (left) illustrates the offline workflow leading to the integration of a trigger table and its deployment in the online validation system. A new configuration is developed in an offline development db where tight development cycles can be carried out. An offline-validated configuration is then migrated to the production database (Online Master Data Storage, OMDS).

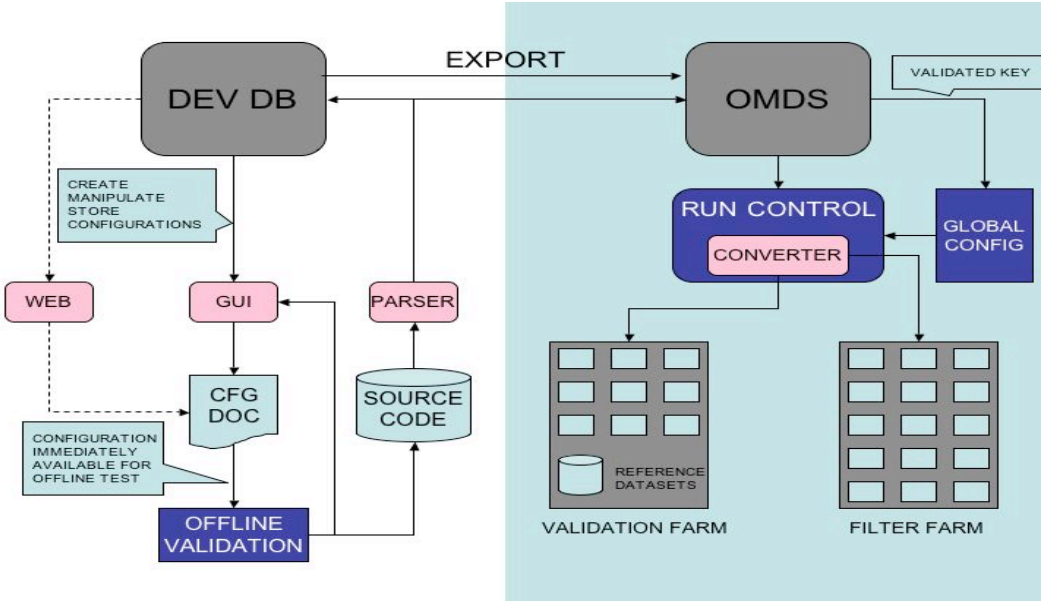

Figure 5. HLT Configuration System workflows

The online workflow starts with the validation of the new configuration in a playback environment (centre), to verify the stability and robustness of new code, and validate trigger algorithms against reference samples. If online validation is successful, the configuration key is stored in the global key database to be used by central Run Control for data taking (right).
The combination of the parser, the GUI, and a private database can also be used to build a convenient development environment for single algorithm developers to test the integration of their newly developed algorithms in a partial or global trigger table.

4.2. Commissioning
The HLT configuration system is currently being commissioned as part of the HLT online validation, using simulated input data and HLT tables produced by the CMS collaboration for offline analysis of the HLT performance [6]. These are realistic tables containing more than one hundred selection paths and making use of a large fraction of the features and algorithms of the CMS reconstruction code.
The system is also in use for data taking from the commissioning of the CMS detector. Although not heavily used to configure complex selection paths, the online workflow is tested under realistic conditions, and the system is proving robust and stable.

5. Summary
The CMS High Level Trigger Configuration Management System is designed to support development, validation, and online operation of the HLT configuration, and is built around the careful design of a database schema to model the features of the underlying reconstruction code. The system is currently in use for commissioning of the CMS detector.

6. Acknowledgments
This work was supported in part by the DOE and NSF (USA), KRF (Korea) and the Marie Curie Program. Prepared in part by LLNL under Contract DE-AC52-07NA27344.

7. References
[1] The CMS Collaboration 1994 *CMS Technical Proposal* (CERN/LHCC 94-38)
[2] The CMS Collaboration 2002 *The TriDAS Project, Technical Design Report, Volume 2: Data Acquisition and High Level Trigger* (CERN/LHCC 2002-36)

[3] Jones C D *et al*. 2007 The New CMS Data Model and Framework *Computing in High Energy Physics (Mumbai, India, 13-17 February 2006)*
[4] Gutleber J and Orsini L 2002 Software Architecture for Processing Clusters Based on i2o *Cluster Computing* **5**(1) 55-64
[5] Bauer G, *et al*. 2007 The Run Control System of the CMS Experiment *These Proceedings*
[6] The CMS Collaboration 2007 *CMS High Level Trigger* (CERN LHCC/2007-021)
Apanasevich L, *et al*. 2007 The CMS High Level Trigger *These Proceedings*